

# Package ‘quanteda’

May 30, 2014

**Type** Package

**Title** Quantitative Analysis of Textual Data

**Version** 0.12

**Date** 2014-04-15

**Author** Ken Benoit and Paul Nulty

**Maintainer** Ken Benoit <kbenoit@lse.ac.uk> and Paul Nulty <p.nulty@lse.ac.uk>

**Description** A library for the quantitative analysis of textual data with R

**Encoding** UTF-8

**License** GPL-3

**Suggests** austin, entropy, jsonlite, openNLP, RJSONIO, RCurl, SnowballC, twitteR, XML

## R topics documented:

bigrams . . . . .	2
clean . . . . .	3
collocations . . . . .	4
corpusAddAttributes . . . . .	5
corpusAppend . . . . .	5
corpusCreate . . . . .	6
corpusFromFileNames . . . . .	6
corpusFromHeaders . . . . .	7
corpusReshape . . . . .	8
countSyllables . . . . .	8
create.fvm.corpus . . . . .	9
describeTexts . . . . .	10
dfm . . . . .	10
dfm2ldaformat . . . . .	12
dfm2tmformat . . . . .	13
dfmTrim . . . . .	13

flatten.dictionary . . . . .	14
getRootFileNames . . . . .	15
getTextDir . . . . .	16
getTextDirGui . . . . .	17
getTextFiles . . . . .	17
getWordStat . . . . .	18
getWordStatCSV . . . . .	18
ieAttribs . . . . .	19
ieTexts . . . . .	19
ieTextsHeaders . . . . .	19
kwic . . . . .	19
kwic2 . . . . .	20
likelihood.test . . . . .	21
MCMCirtPoisson1d . . . . .	22
movies . . . . .	24
naiveBayesText . . . . .	24
ngrams . . . . .	25
predict.naivebayes . . . . .	26
readWStatDict . . . . .	27
removeStopwords . . . . .	27
selectFeatures . . . . .	28
sentenceSeg . . . . .	29
stopwords_EN . . . . .	30
subset.corpus . . . . .	30
summary.corpus . . . . .	31
swissdebates . . . . .	31
sylCounts . . . . .	32
tagPos . . . . .	32
tokenize . . . . .	33
translate . . . . .	33
translate.corpus . . . . .	34
twitterSearch . . . . .	35
twitterStreamer . . . . .	35
twitterTerms . . . . .	35
wordfishMCMC . . . . .	36
<b>Index</b>	<b>39</b>

---

bigrams	<i>Create bigrams</i>
---------	-----------------------

---

**Description**

Create bigrams

**Usage**

bigrams(text, window = 1)

**Arguments**

text	character vector containing the texts from which ngrams will be extracted
window	how many words to be counted for adjacency. Default is 1 for only immediately neighbouring words.

**Value**

a character of bigrams vector

**Author(s)**

Kohei Watanabe and Ken Benoit

**Examples**

```
data(iebudgets)
bigrams("The quick brown fox jumped over the lazy dog.")
bigrams("The quick brown fox jumped over the lazy dog.", window=2)
```

---

clean	<i>Perform basic cleanup on a character object</i>
-------	--

---

**Description**

Simple cleanup for strings, removing punctuation, converting to lowercase and optionally replacing some language-specific characters

**Usage**

```
clean(s, langNorm = FALSE, removeDigits = TRUE, lower = TRUE,
      removePunct = TRUE)
```

**Arguments**

s	character object to be cleaned
langNorm	If true, French and German special characters are normalized.
removeDigits	If true, digits are removed. Default is TRUE
lower	If true, string is converted to lowercase. Default is TRUE

**Value**

character object in lowercase with punctuation (and optionally digits) removed

**Examples**

```
## Not run:
s <- "A cursed f$&^! Exclamation! point; paragraph §1.2, which I wrote."
clean(s)

## End(Not run)
```

---

collocations	<i>Detect collocations in a text</i>
--------------	--------------------------------------

---

**Description**

returns a list of collocations. Note: Currently works only for pairs (bigram collocations).

**Usage**

```
collocations(text = NULL, file = NULL, top = NA, distance = 2, n = 2,
  method = c("lr", "chi2", "mi"))
```

**Arguments**

text	a text or vector of texts
file	a filename containing a text
top	threshold number for number of collocations to be returned (in descending order of association value)
distance	distance between pairs of collocations
method	association measure for detecting collocations

**Value**

A list of collocations, their frequencies, and their test statistics

**Author(s)**

Kenneth Benoit

**Examples**

```
data(iebudgets)
collocations(iebudgets$attribs$texts[1], top=50)
collocations(iebudgets$attribs$texts[1], top=50, method="chi2")
```

---

corpusAddAttributes	<i>This function adds a named list of attributes to an existing corpus</i>
---------------------	--

---

**Description**

This function adds a named list of attributes to an existing corpus

**Usage**

```
corpusAddAttributes(corpus, newattribs, name = newattribs)
```

**Arguments**

corpus	Corpus to add attributes to
newattribs	A list of new attributes should be a named list of length(corpus\$texts)
name	A name for the new attributes

**Value**

corpus A corpus with the new attributes added

---

corpusAppend	<i>function to add new texts and attributes to an existing corpus Accepts a list of texts and a list of associated attributes and adds them to the corpus</i>
--------------	---

---

**Description**

function to add new texts and attributes to an existing corpus Accepts a list of texts and a list of associated attributes and adds them to the corpus

**Usage**

```
corpusAppend(corpus1, newtexts, newattribs, ...)
```

**Arguments**

corpus1	An existing corpus to add new texts and attributes to
newtexts	New texts to be added to the corpus
newattribs	New attribs associated with the new texts text

**Examples**

```
data(iebudgets)
data(ieAttribs)
data(ieTexts)
budgets <- corpusAppend(iebudgets, ieTexts, ieAttribs)
```

---

corpusCreate	<i>Create a new corpus This function creates a corpus from a character vector (of texts), adds text-specific variables (which we term "attributes"), along with optional meta-data and notes.</i>
--------------	---

---

### Description

Create a new corpus This function creates a corpus from a character vector (of texts), adds text-specific variables (which we term "attributes"), along with optional meta-data and notes.

### Usage

```
corpusCreate(texts, attribs = NULL, textnames = NULL, source = NULL,
             notes = NULL)
```

### Arguments

texts	A character vector containing the texts to be stored in the corpus.
textnames	Names to be assigned to the texts, defaults to the names of the character vector (if any), otherwise assigns "text1", "text2", etc.
attribs	A data frame of attributes that is associated with each text.
source	A string specifying the source of the texts, used for referencing.
notes	A string containing notes about who created the text, warnings, To Dos, etc.

### Examples

```
data(ieTexts)
data(ieAttribs)
budgets <- corpusCreate(ieTexts, attribs=ieAttribs)
summary(budgets)
```

---

corpusFromFilenames	<i>create a new corpus with attribute-value pairs taken from filenames</i>
---------------------	--

---

### Description

This function takes a directory, reads in all the documents in that directory and makes a new corpus where the attributes and values are created by splitting the filename according to a separator. For example, a directory may contain files with a naming scheme that identifies attribute values, e.g.: "2010\_BUDGET\_05\_Brian\_Cowen\_FF.txt".

### Usage

```
corpusFromFilenames(directory, attNames, sep = "_")
```

**Arguments**

directory	Path to folder containing documents
attNames	A vector naming the attribute types
sep	A string by which the filename should be separated to get the values. Default is underscore.

**Details**

To create a corpus object from texts named in this format, we can call this function and specify the attribute types and separator, e.g: `new_corpus <- corpusFromFilenames(dirname, c("country", "electionType", "year", "language", "party"), sep='_')`

Underscore is the default separator

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:
new_corpus <- corpusFromFilenames(dirname, c("country", "electionType", "year", "language", "party"), sep='_')

## End(Not run)
```

---

corpusFromHeaders	<i>create a new corpus with attribute-value pairs taken from document headers</i>
-------------------	---

---

**Description**

This function takes a vector of texts with JSON headers and makes a new corpus where the attributes and values are created from JSON headers in the text. The JSON header should be the first line (as delimited by \n) in document. For example, a document may begin as follows: "budgetPosition" : "1.0", "party": "FF"} When I presented the supplementary budget to this House last April....

**Usage**

```
corpusFromHeaders(headerTexts)
```

**Arguments**

headerTexts	A vector of texts with JSON headers
-------------	-------------------------------------

**Details**

The directory must contain only documents to be used in the corpus, and each document must have the same attributes.

**Author(s)**

Paul Nulty

**Examples**

```
data(ieTextsHeaders)
budgets <- corpusFromHeaders(ieTextsHeaders)
```

---

corpusReshape	<i>Transform a corpus by splitting texts into sentences</i>
---------------	---

---

**Description**

Each text in the corpus is split into sentences, and each sentence becomes a standalone text, with attributes indicating the text it is taken from and it's serial number in that text

**Usage**

```
corpusReshape(corpus)
```

**Arguments**

corpus	Corpus to transform
feature	Feature to count

**Examples**

```
## Not run:
corpus <- data(iebudgets)
sentCorp <- corpus.reshape(corpus)

## End(Not run)
```

---

countSyllables	<i>Returns a count of the number of syllables in the input This function takes a text and returns a count of the number of syllables it contains. For British English words, the syllable count is exact and looked up from the CMU pronunciation dictionary. For any word not in the dictionary the syllable count is estimated by counting vowel clusters.</i>
----------------	--

---

**Description**

Returns a count of the number of syllables in the input This function takes a text and returns a count of the number of syllables it contains. For British English words, the syllable count is exact and looked up from the CMU pronunciation dictionary. For any word not in the dictionary the syllable count is estimated by counting vowel clusters.



**Usage**

```
countSyllables(sourceText, verbose = FALSE)
```

**Arguments**

sourceText      Text to be counted

**Value**

numeric A count (estimate) of the number of syllables in sourceText

**Examples**

```
countSyllables("This is an example sentence.")
```

---

create.fvm.corpus	<i>Create a feature-value matrix from a corpus object returns a feature value matrix compatible with austin</i>
-------------------	---

---

**Description**

Create a feature-value matrix from a corpus object returns a feature value matrix compatible with austin

**Usage**

```
create.fvm.corpus(corpus, feature = c("word"), stem = FALSE,
  remove_stopwords = FALSE, groups = NULL, subset = NULL,
  verbose = TRUE)
```

**Arguments**

corpus            Corpus to make matrix from  
 feature          Feature to count  
 feature          type to aggregate by, default is file

**Examples**

```
## Not run:
fvm <- create.fvm.corpus(budgets, group="party")

## End(Not run)
```

---

describeTexts	<i>print a summary of texts Prints to the console a description of the texts, including number of types, tokens, and sentences</i>
---------------	--

---

## Description

print a summary of texts Prints to the console a description of the texts, including number of types, tokens, and sentences

## Usage

```
describeTexts(texts, verbose = TRUE)
```

## Arguments

texts	The texts to be described
-------	---------------------------

## Examples

```
texts <- c("testing this text", "and this one")
describeTexts(texts)
```

---

dfm	<i>Create a document-feature matrix from a corpus object</i>
-----	--

---

## Description

returns a document by feature matrix compatible with austin. A typical usage would be to produce a word-frequency matrix where the cells are counts of words by document.

## Usage

```
dfm(corpus, feature = c("word"), stem = FALSE, stopwords = NULL,
    bigram = FALSE, groups = NULL, subset = NULL, verbose = TRUE,
    dictionary = NULL, dictionary.regex = FALSE, addto = NULL)
```

```
## S3 method for class 'corpus'
```

```
dfm(corpus, feature = c("word"), stem = FALSE,
    stopwords = NULL, bigram = FALSE, groups = NULL, subset = NULL,
    verbose = TRUE, dictionary = NULL, dictionary.regex = FALSE,
    addto = NULL)
```

**Arguments**

corpus	Corpus from which to generate the document-feature matrix
feature	Feature to count (e.g. words)
stem	Stem the words
stopwords	A character vector of stopwords that will be removed from the text when constructing the <a href="#">dfm</a> . If NULL (default) then no stopwords will be applied. If "TRUE" then it currently defaults to <a href="#">stopwords_EN</a> .
groups	Grouping variable for aggregating documents
subset	Expression for subsetting the corpus before processing
verbose	Get info to screen on the progress
dictionary	A list of character vector dictionary entries, including regular expressions (see examples)
dictionary.regex	TRUE means the dictionary is already in regular expression format, otherwise it will be converted from "wildcard" format
addto	NULL by default, but if an existing dfm object is specified, then the new dfm will be added to the one named. If both dfms are built from dictionaries, the combined dfm will have its Non_Dictionary total adjusted.

**Value**

A matrix object with row names equal to the document names and column names equal to the feature labels. This matrix has `names(dimnames) = c("docs", "words")` to make it conformable to an [wfm](#) object.

**Author(s)**

Kenneth Benoit

**Examples**

```
data(iebudgets)
wfm <- dfm(iebudgets)

## by party, subset for 2010
wfmByParty2010 <- dfm(subset(iebudgets, year==2010), groups="party")

## with dictionaries
corpus <- subset(iebudgets, year==2010)
mydict <- list(christmas=c("Christmas", "Santa", "holiday"),
               opposition=c("Opposition", "reject", "notincorpus"),
               taxing="taxing",
               taxation="taxation",
               taxregex="tax*")
dictDfm <- dfm(corpus, dictionary=mydict)
dictDfm
```

```
## removing stopwords
testText <- "The quick brown fox named Séamus jumps over the lazy dog Rory, with Tom's newspaper in his mouth."#
testCorpus <- corpusCreate(testText)
dfm(testCorpus, stopwords=TRUE)
if (require(tm)) {
}

## adding one dfm to another
mydict2 <- list(partyref=c("Lenihan", "Fianna", "Sinn", "Gael"))
dictDfm2 <- dfm(corpus, dictionary=mydict2, addto=dictDfm)
dictDfm2
```

---

dfm2ldaformat	<i>Convert a quanteda <a href="#">dfm</a> (document feature matrix) into a the data format needed by <a href="#">lda</a></i>
---------------	--

---

## Description

Convert a quanteda [dfm](#) (document feature matrix) into a the data format needed by [lda](#)

## Usage

```
dfm2ldaformat(d)
```

## Arguments

d                      A [dfm](#) object

## Value

A list with components "documents" and "vocab" as needed by [lda.collapsed.gibbs.sampler](#)

## Examples

```
## Not run:
data(iebudgets)
iebudgets2010 <- subset(iebudgets, year==2010)
# create document-feature matrix, remove stopwords
d <- dfm(iebudgets2010, stopwords=TRUE)
# trim low frequency words
d <- dfmTrim(d, minCount=5, minDoc=3)
td <- dfm2ldaformat(d)
if (require(lda)) {
  tmodel.lda <- result <- lda.collapsed.gibbs.sampler(documents=td$documents,
                                                    K=10,
                                                    vocab=td$vocab,
                                                    num.iterations=50, alpha=0.1, eta=0.1)
}
top.topic.words(tmodel.lda$topics, 10, by.score=TRUE) # top five words in each topic

## End(Not run)
```

---

dfm2tmformat	<i>Convert a quanteda <a href="#">dfm</a> (document feature matrix) into a <b>tm</b> <a href="#">DocumentTermMatrix</a></i>
--------------	---

---

### Description

**tm** represents sparse document-feature matrixes in the [simple triplet matrix](#) format of the package **slam**. This function converts a [dfm](#) into a [DocumentTermMatrix](#), for working with the [dfm](#) in **tm** or in other packages that expect this format, such as **topicmodels**.

### Usage

```
dfm2tmformat(d, weighting = weightTf, ...)
```

### Arguments

d	A <a href="#">dfm</a> object
weighting	<b>tm</b> 's coercion function accepts weightings such as tf-idf, see <b>tm</b> 's <a href="#">as.DocumentTermMatrix</a> for a list of possible arguments. The default is just tf (term frequency)

### Value

A simple triplet matrix of class [as.DocumentTermMatrix](#)

### Examples

```
data(iebudgets)
iebudgets2010 <- subset(iebudgets, year==2010)
d <- dfmTrim(dfm(iebudgets2010), minCount=5, minDoc=3)
dim(d)
td <- dfm2tmformat(d)
length(td$v)
if (require(topicmodels)) tmodel.lda <- LDA(td, control = list(alpha = 0.1), k = 4)
```

---

dfmTrim	<i>Trim a <a href="#">dfm</a> based on a subset of features and words</i>
---------	---

---

### Description

Returns a document by feature matrix reduced in size based on document and term frequency, and/or subsampling.

### Usage

```
dfmTrim(dfm, minCount = 5, minDoc = 5, sample = NULL, verbose = TRUE)
```

**Arguments**

dfm	Document-feature matrix created by <a href="#">dfm</a>
minCount	minimum feature count
minDoc	minimum number of documents in which a feature appears
sample	how many features to retain (based on random selection)
verbose	print messages

**Value**

A dfm matrix object reduced in size.

**Author(s)**

Will Lowe, adapted by Ken Benoit

**Examples**

```
data(iebudgets)
dtm <- dfm(iebudgets)
dim(dtm) # 196 docs x 13343 words
dtmReduced <- dfmTrim(dtm, minCount=10, minDoc=3) # only words occurring at least 10 times and in at least 3 documents
dim(dtmReduced) # 196 docs x 3006 words
dtmSampled <- dfmTrim(dtm, sample=200) # top 200 words
dim(dtmSampled) # 196 x 200 words
```

---

flatten.dictionary	<i>Flatten a hierarchical dictionary into a list of character vectors</i>
--------------------	---

---

**Description**

Converts a hierarchical dictionary (a named list of named lists, ending in character vectors at the lowest level) into a flat list of character vectors. Works like `unlist(dictionary, recursive=TRUE)` except that the recursion does not go to the bottom level.

**Usage**

```
flatten.dictionary(elms, parent = "", dict = list())
```

**Arguments**

elms	list to be flattened
parent	parent list name, gets built up through recursion in the same way that <code>unlist(dictionary, recursive=TRUE)</code> works
dict	the bottom list of dictionary entries ("synonyms") passed up from recursive calls

**Details**

Called by dfm()

**Value**

A dictionary flattened down one level further than the one passed

**Author(s)**

Kohei Watanabe

**Examples**

```
dictPopulismEN <-
  list(populism=c("elit*", "consensus*", "undemocratic*", "referend*",
    "corrupt*", "propagand", "politici*", "*deceit*",
    "*deceiv*", "*betray*", "shame*", "scandal*", "truth*",
    "dishonest*", "establishm*", "ruling*"))
flatten.dictionary(dictPopulismEN)

hdict <- list(level1a = list(level1a1 = c("l1a11", "l1a12"),
  level1a2 = c("l1a21", "l1a22")),
  level1b = list(level1b1 = c("l1b11", "l1b12"),
    level1b2 = c("l1b21", "l1b22", "l1b23")),
  level1c = list(level1c1a = list(level1c1a1 = c("lowest1", "lowest2")),
    level1c1b = list(level1c1b1 = c("lowestalone"))))
flatten.dictionary(hdict)
```

---

getRootFileNames	<i>Truncate absolute filepaths to root filenames</i>
------------------	--

---

**Description**

This function takes an absolute filepath and returns just the document name

**Usage**

```
getRootFileNames(longFileNames)
```

**Arguments**

longFileNames Absolute filenames including a full path with directory

**Value**

character vector of filenames withouth directory path

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:  
getRootFileNames('/home/paul/documents/libdem09.txt')  
  
## End(Not run)
```

---

getTextDir	<i>loads all text files from a given directory</i>
------------	--

---

**Description**

given a directory name, get a list of all files in that directory and load them into a character vector using getTextFiles

**Usage**

```
getTextDir(dirname)
```

**Arguments**

dirname	A directory path
---------	------------------

**Value**

character vector of texts read from disk

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:  
getTextDir('/home/paul/documents/')  
  
## End(Not run)
```



---

getTextDirGui	<i>provides a gui interface to choose a gui to load texts from</i>
---------------	--

---

**Description**

launches a GUI to allow the user to choose a directory from which to load all files.

**Usage**

```
getTextDirGui()
```

**Value**

character vector of texts read from disk

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:  
getTextFiles('/home/paul/documents/libdem09.txt')  
  
## End(Not run)
```

---

getTextFiles	<i>load text files from disk into a vector of character vectors points to files, reads them into a character vector of the texts with optional names, default being filenames returns a named vector of complete, unedited texts</i>
--------------	--

---

**Description**

load text files from disk into a vector of character vectors points to files, reads them into a character vector of the texts with optional names, default being filenames returns a named vector of complete, unedited texts

**Usage**

```
getTextFiles(filenames, textnames = NULL, verbose = FALSE)
```

**Arguments**

filenames	a vector of paths to text files
textnames	names to assign to the texts

**Value**

character vector of texts read from disk

**Author(s)**

Paul Nulty

**Examples**

```
## Not run:
getTextFiles('/home/paul/documents/libdem09.txt')

## End(Not run)
```

---

getWordStat

*Imports a Wordstat corpus from an XML file*


---

**Description**

Reads in a wordstat XML file and creates a corpus object with the document as text and variables as attributes

**Usage**

```
getWordStat(filename = NULL)
```

**Arguments**

filename            Path to wordstat XML file

---

getWordStatCSV

*Imports a Wordstat corpus from a CSV file*


---

**Description**

Reads in a wordstat CSV file and creates a corpus object with the document as text and variables as attributes

**Usage**

```
getWordStatCSV(filename = NULL)
```

**Arguments**

filename            Path to wordstat CSV file

---

ieAttribs	<i>A vector of attributes to match ieBudget documents</i>
-----------	---

---

**Description**

This is a small vector of attributes for use in examples with ieBudgets

---

ieTexts	<i>A vector of texts from Irish budget debates for use with examples</i>
---------	--

---

**Description**

This is a small vector of texts from the ieBudget corpus for use with testing examples

This is a small vector of texts from the ieBudget corpus for use with testing examples

---

ieTextsHeaders	<i>A version of ieTexts with attributes contained in JSON headers</i>
----------------	---

---

**Description**

This is a small vector of texts for use in examples with corpusFromHeaders

---

kwic	<i>List key words in context from a text or a corpus of texts.</i>
------	--

---

**Description**

For a text or a collection of texts (in a quanteda corpus object), return a list of a keyword supplied by the user in its immediate context, identifying the source text and the word index number within the source text. (Not the line number, since the text may or may not be segmented using end-of-line delimiters.)

**Usage**

```
kwic(text, word, window = 5)
```

```
## S3 method for class 'character'
kwic(text, word, window = 5)
```

```
## S3 method for class 'corpus'
kwic(corpus, word, window = 5)
```

**Arguments**

text	A text character scalar or a quanteda corpus. (Currently does not support character vectors.)
word	A keyword chosen by the user.
window	The number of context words to be displayed around the keyword.
text	a text character scalar (Currently does not support character vectors.)
corpus	a quanteda corpus object

**Value**

A data frame with the context before (preword), the keyword in its original format (word, preserving case and attached punctuation), and the context after (postword). The rows of the dataframe will be named with the word index position, or the text name and the index position for a corpus object.

**Author(s)**

Kenneth Benoit and Paul Nulty

**Examples**

```
data(iebudgets)
kwic(subset(iebudgets, year==2010), "Christmas", window=4)
```

---

kwic2

---

*This function is an alternative KWIC*


---

**Description**

This function is an alternative KWIC

**Usage**

```
kwic2(texts, word, window = 30, filter = "", location = TRUE,
      case = TRUE)
```

**Arguments**

text	Texts
word	Word of interest
window	Window span in character
filter	Filter files in texts by regular expression
location	Show location of the word
case	Ignore case

**Value**

cfvm2 Collocatons as data frame

**Author(s)**

Kohei Watanabent

**Examples**

```
## Not run:
kwic2(texts, "we", filter = '_2010', location=TRUE)

## End(Not run)
```

---

likelihood.test	<i>likelihood test for 2x2 tables</i>
-----------------	---------------------------------------

---

**Description**

returns a list of values

**Usage**

```
likelihood.test(x)
```

**Arguments**

x                      a contingency table or matrix object

**Value**

A list of return values

**Author(s)**

Kenneth Benoit

---

MCMCirtPoisson1d	<i>Bayesian-MCMC version of a 1-dimensional Poisson IRT scaling model</i>
------------------	---

---

## Description

MCMCirtPoisson1d implements a flexible, Bayesian model estimated in JAGS using MCMC. It is based on the implementation of [wordfish](#) from the [austin](#) package. Options include specifying a model for alpha using document-level covariates, and partitioning the word parameters into different subsets, for instance, countries.

## Usage

```
MCMCirtPoisson1d(dtm, dir = c(1, 2), control = list(sigma = 3, startparams =
  NULL), verbose = TRUE, itembase = 1, startRandom = FALSE, nChains = 1,
  nAdapt = 100, nUpdate = 300, nSamples = 200, nThin = 1, ...)
```

## Arguments

dtm	The document-term matrix. Ideally, documents form the rows of this matrix and words the columns, although it should be correctly coerced into the correct shape.
dir	A two-element vector, enforcing direction constraints on theta and beta, which ensure that $\theta_{\text{dir}[1]} < \theta_{\text{dir}[2]}$ . The elements of dir will index documents.
control	list specifies options for the estimation process. These are: tol, the proportional change in log likelihood sufficient to halt estimation, sigma the standard deviation for the beta prior in poisson form, and startparams a previously fitted wordfish model. verbose generates a running commentary during estimation. See <a href="#">wordfish</a> .
itembase	A index or column name from dtm indicating which item should be used as the reference category. (These will have $\beta_j = 0$ and $\alpha_j = 0$ .) The default is 1, to use the first category. If set to NULL then no constraints will be implemented. See details.
verbose	Turn this on for messages. Default is TRUE.
startRandom	FALSE by default, uses random starting values (good for multiple chains) if TRUE
nChains	Number of chains to run in JAGS.
nAdapt	Adaptation iterations in JAGS.
nUpdate	Update iterations in JAGS.
nSamples	Number of posterior samples to draw in JAGS.
nThin	Thinning parameter for drawing posterior samples in JAGS.
...	Additional arguments passed through.

## Details

The ability to constrain an item is designed to make the additive Poisson GLM mathematically equivalent to the multinomial model for  $R \times C$  contingency tables. We recommend setting a neutral category to have  $\psi_0 = 0$  and  $\beta_0 = 0$ , for example the word "the" for a text count model (assuming this word has not been removed). Note: Currently the item-level return values will be returned in the original order supplied (psi and beta) but this is not true yet for the `mcmc.samples` value, which will have the constrained category as index 1. (We will fix this soon.)

## Value

An augmented `wordfish` class object with additional stuff packed in. To be documented.

## Author(s)

Kenneth Benoit

## Examples

```
## Not run:
data(iebudgets)
# extract just the 2010 debates
iebudgets2010 <- subset(iebudgets, year==2010)

# create a document-term matrix and set the word margin to the columns
dtm <- dfm(iebudgets2010)

# estimate the maximum likelihood wordfish model from austin
require(austin)
iebudgets2010_wordfish <- wordfish(as.wfm(dtm, word.margin=2), dir=c(2,1))

# estimate the MCMC model, default values
iebudgets2010_wordfishMCMC <- MCMCirtPoisson1d(dtm, itembase="the", dir=c(2,1))
iebudgets2010_wordfishMCMC_unconstrained <- MCMCirtPoisson1d(dtm, dir=c(2,1))

# compare the estimates of  $\theta_i$ 
require(psych)
pairs.panels(data.frame(ML=iebudgets2010_wordfish$theta,
                        PoissonThe=iebudgets2010_wordfishMCMC$theta,
                        PoissonUnconst=iebudgets2010_wordfishMCMC_unconstrained$theta),
             smooth=FALSE, scale=FALSE, ellipses=FALSE, lm=TRUE, cex.cor=2.5)

# inspect a known "opposition" word beta values
iebudgets2010_wordfish$beta[which(iebudgets2010_wordfishMCMC_unconstrained$words=="fianna")]
iebudgets2010_wordfishMCMC$beta[which(iebudgets2010_wordfishMCMC_unconstrained$words=="fianna")]
iebudgets2010_wordfishMCMC_unconstrained$beta[which(iebudgets2010_wordfishMCMC_unconstrained$words=="fianna")]

# random starting values, for three chains
dtm.sample <- trim(dtm, sample=200)
iebudgets2010_wordfishMCMC_sample <- MCMCirtPoisson1d(dtm.sample, dir=c(2,1), startRandom=TRUE, nChains=3)

## End(Not run)
```

---

movies	<i>A corpus object containing 2000 movie reviews</i>
--------	--

---

**Description**

A corpus object containing 2000 movie reviews classified by positive or negative sentiment

**References**

<http://dl.acm.org/citation.cfm?id=1118704>

---

naiveBayesText	<i>Naive Bayes classifier for texts</i>
----------------	---

---

**Description**

Naive Bayes classifier for texts

**Usage**

```
naiveBayesText(x, y, smooth = 1, prior = "uniform",  
               distribution = "multinomial", ...)
```

**Arguments**

x	character vector of training texts
y	character vector of test texts
smooth	smoothing parameter for feature counts by class
prior	prior distribution on texts, see details
distribution	count model for text features, can be multinomial or Bernoulli
...	

**Details**

Currently working for vectors of texts.



**Value**

A list of return values, consisting of:

call	original function call
PwGc	probability of the word given the class (empirical likelihood)
Pc	class prior probability
PcGw	posterior class probability given the word
Pw	baseline probability of the word
data	list consisting of x training class, and y test class
distribution	the distribution argument
prior	argument passed as a prior
smooth	smoothing parameter

**Author(s)**

Kenneth Benoit

---

ngrams

*Create ngrams*

---

**Description**

Create a set of ngrams (words in sequence) from a text.

**Usage**

```
ngrams(text, n = 2, concatenator = "_")
```

**Arguments**

text	character vector containing the texts from which ngrams will be extracted
n	the number of tokens to concatenate. Default is 2 for bigrams.
window	how many words to be counted for adjacency. Default is 1 for only immediately neighbouring words.
concatenator	character for combining words, default is _ (underscore) character

**Value**

a character of bigrams vector

**Author(s)**

Kohei Watanabe, Ken Benoit, Paul Nulty

**Examples**

```
data(iebudgets)
ngrams("The quick brown fox jumped over the lazy dog.", n=2)
ngrams("The quick brown fox jumped over the lazy dog.", n=3)
```

---

predict.naivebayes	<i>prediction method for Naive Bayes classifiers</i>
--------------------	--

---

**Description**

prediction method for Naive Bayes classifier objects

**Usage**

```
## S3 method for class 'naivebayes'
predict(object, newdata = NULL, scores = c(-1, 1))
```

**Arguments**

object	a naivebayes class object
newdata	new data on which to perform classification
scores	"reference" values when the wordscores equivalent implementation of Naive Bayes prediction is used. Default is c(-1, 1).

**Details**

implements class predictions using trained Naive Bayes examples (from naiveBayesText())

**Value**

A list of two data frames, named docs and words corresponding to word- and document-level predicted quantities

docs	data frame with document-level predictive quantities: nb.predicted, ws.predicted, bs.predicted, PcGw, wordscore.doc, bayesscore.doc, posterior.diff, posterior.logdiff. Note that the diff quantities are currently implemented only for two-class solutions.
words	data-frame with word-level predictive quantities: wordscore.word, bayesscore.word

**Author(s)**

Kenneth Benoit

---

readWStatDict	<i>This function makes a flattened dictionary from a Wordstat hierarchical dictionary</i>
---------------	---

---

**Description**

This function makes a flattened dictionary from a Wordstat hierarchical dictionary

**Usage**

```
readWStatDict(path)
```

**Author(s)**

Paul Nulty

---

removeStopwords	<i>remove common or 'semantically empty' words from a text. This function takes a character vector 'text' and removes words in the list provided in 'stopwords'. If no list of stopwords is provided a default list for English is used.</i>
-----------------	--

---

**Description**

remove common or 'semantically empty' words from a text. This function takes a character vector 'text' and removes words in the list provided in 'stopwords'. If no list of stopwords is provided a default list for English is used.

**Usage**

```
removeStopwords(text, stopwords = NULL)
```

**Arguments**

text	Text to alter
stopwords	List of stopwords to remove

**Value**

a character vector of text with stopwords removed

**Examples**

```
someText <- "Here is an example of text containing some stopwords we want to remove."
removeStopwords(someText)
removeStopwords(someText, stopwords=c("is", "an", "to"))
```

---

selectFeatures	<i>extract feature words This function takes type of feature extractor and a word frequency matrix with binary class (1/0) to select features in class one. 'wsll' and 'wschisq' replicates of 'Keyness' of Wordsmith Tools.</i>
----------------	--

---

### Description

extract feature words This function takes type of feature extractor and a word frequency matrix with binary class (1/0) to select features in class one. 'wsll' and 'wschisq' replicates of 'Keyness' of Wordsmith Tools.

extract feature words This function takes type of feature extractor and a word frequency matrix with binary class (1/0) to select features in class one. 'wsll' and 'wschisq' replicates of 'Keyness' of Wordsmith Tools.

### Usage

```
selectFeatures(extractor, dfm, class, smooth = 1, show = 10)
```

```
selectFeatures(extractor, dfm, class, smooth = 1, show = 10)
```

### Arguments

extractor	Type of feature extractor
dfm	Word frequency matrix
class	Binary class
smooth	Smoothing constant
show	Number of features shown
extractor	Type of feature extractor
dfm	Word frequency matrix
class	Binary class
smooth	Smoothing constant
show	Number of features shown

### Value

data frame of feature words

data frame of feature words

### Author(s)

Kohei Watanabe

Kohei Watanabe

## Examples

```
## Not run:
texts <- getTextDir("/home/kohei/Documents/budget_2010/")
class <- rep(0, length(texts))
class[grepl("_LAB", names(texts))] <- 1
class[grepl("_FF", names(texts))] <- 0
corpus <- corpusCreate(texts, attribs=list(class=class))
dfm <- dfm(corpus)
features <- selectFeatures('ll', dfm, corpus$attribs$class, smooth=1)

## End(Not run)
## Not run:
texts <- getTextDir("/home/kohei/Documents/budget_2010/")
class <- rep(0, length(texts))
class[grepl("_LAB", names(texts))] <- 1
class[grepl("_FF", names(texts))] <- 0
corpus <- corpusCreate(texts, attribs=list(class=class))
dfm <- dfm(corpus)
features <- selectFeatures('ll', dfm, corpus$attribs$class, smooth=1)

## End(Not run)
```

---

sentenceSeg	<i>split a text into sentences This function takes a text and splits it into sentences.</i>
-------------	---

---

## Description

split a text into sentences This function takes a text and splits it into sentences.

## Usage

```
sentenceSeg(text, pat = "[\\.\n?\\!][\\n* ]|\\n\\n*",
  abbreviations = NULL)
```

## Arguments

text	Text to be segmented
abbreviations	A list of abbreviations'.' and therefore should not be used to segment text

## Examples

```
test <- "This is a sentence! Several sentences. It's designed by a Dr. to test whether this function works. Or not"
sentenceSeg(test)
```

---

stopwords_EN	<i>A list containing common English stopwords</i>
--------------	---

---

### Description

A list containing common English stopwords

---

subset.corpus	<i>extract a subset of a corpus</i>
---------------	-------------------------------------

---

### Description

Works just like the normal subset command but for corpus objects

### Usage

```
## S3 method for class 'corpus'
subset(corpus, subset = NULL, select = NULL)
```

### Arguments

corpus	corpus object to be subsetted.
subset	logical expression indicating elements or rows to keep: missing values are taken as false.
select	expression, indicating the attributes to select from the corpus

### Value

corpus object

### Examples

```
## Not run:
data(iebudgets)
iebudgets2010 <- subset(iebudgets, year==2010)
summary(iebudgets2010)
iebudgetsLenihan <- subset(iebudgets, speaker="Lenihan", select=c(speaker, year))
summary(iebudgetsLenihan)

## End(Not run)
```

---

summary.corpus	<i>Corpus summary</i>
----------------	-----------------------

---

### Description

Displays information about a corpus object, including attributes and metadata such as date of number of texts, creation and source.

### Usage

```
## S3 method for class 'corpus'
summary(corpus, nmax = 100, texts = "texts",
        subset = NULL)
```

### Arguments

corpus	An existing corpus to be summarized
nmax	maximum number of texts to describe, default=100
texts	The name of the attribute containing the corpus texts, if not 'texts'. For instance, if the corpus contained translated texts as an attribute, then setting this to the name of that variable would make it possible to summarize the alternate rather than the main texts.
subset	a Boolean expression that specifies a subset of the texts, similar to subset.corpus

### Examples

```
data(iebudgets)
summary(iebudgets, subset=(year==2010))
summary(iebudgets, nmax=10)
```

---

swissdebates	<i>A corpus object containing 770 texts from debates in the Swiss parliament</i>
--------------	--

---

### Description

A corpus object containing 770 texts from debates in the Swiss parliament

### References

<http://dl.acm.org/citation.cfm?id=1118704>

---

sylCounts	<i>A named list mapping words to counts of their syllables</i>
-----------	--

---

### Description

A named list mapping words to counts of their syllables, generated from the CMU pronunciation dictionary

### References

<http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

---

tagPos	<i>Returns a table of the occurrences of different parts of speech in a sentence This function takes a sentence and tags each word with it's part of speech using openNLP's POS tagger, then returns a table of the parts of speech</i>
--------	---

---

### Description

[http://www.ling.upenn.edu/courses/Fall\\_2003/ling001/penn\\_treebank\\_pos.html](http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)

### Usage

```
tagPos(sentence)
```

### Arguments

sentence	Sentence to be tagged
----------	-----------------------

### Examples

```
## Not run:  
tagPos("This is an example sentence with nouns and verbs for tagging.")  
  
## End(Not run)
```



---

tokenize	<i>Split a string into words The input text is split into words by whitespace</i>
----------	---

---

**Description**

Split a string into words The input text is split into words by whitespace

**Usage**

```
tokenize(str, langNorm = FALSE, removeDigits = TRUE, lower = TRUE,
  removePunct = TRUE)
```

**Arguments**

str	String to be tokenized
langNorm	If TRUE (default), French and German special characters are normalized
removeDigits	If TRUE (default), digits are removed
lower	If TRUE (default), string is converted to lowercase
removePunct	If TRUE (default), punctuation is removed

**Value**

a character vector containing the input text tokens

**Examples**

```
testtxt <- "The quick brown fox named Séamus jumps over the lazy dog Rory, with Tom's newspaper in his mouth."
tokenize(testtxt)
tokenize(testtxt, lower=FALSE)
```

---

translate	<i>Send text to the google translate research API This function translates a text by sending it to the google translate API.</i>
-----------	--

---

**Description**

Send text to the google translate research API This function translates a text by sending it to the google translate API.

**Usage**

```
translate(sourceText, sourceLanguage, targetLanguage, key = NULL,
  verbose = FALSE)
```

**Arguments**

sourceText      Text to be translated

sourceLanguage   Language of the source text

targetLanguage   Language of the translated text

key                API key for Google Translate research API

**Examples**

```
## Not run: translation <- translate(original, fr, de, key='insertkeyhere')
```

---

translate.corpus	<i>Send a corpus to the google translate research API This function translates a the texts in a corpus by sending them to the google translate API.</i>
------------------	---

---

**Description**

Send a corpus to the google translate research API This function translates a the texts in a corpus by sending them to the google translate API.

**Usage**

```
translate.corpus(corpus, targetlanguageString, textvar = "texts",
  languagevar = "language", key = NULL)
```

**Arguments**

corpus                corpus to be translated

targetlanguageString      Language of the source text

languagevar          Language of the translated text

**Examples**

```
## Not run:
translation <- translate(original, fr, de, key='insertkeyhere')

## End(Not run)
```

---

twitterSearch	<i>work-in-progress from-scratch interface to Twitter search API</i>
---------------	--

---

**Description**

work-in-progress from-scratch interface to Twitter search API

**Usage**

```
twitterSearch()
```

---

twitterStreamer	<i>work-in-progress interface to Twitter streaming API</i>
-----------------	--

---

**Description**

work-in-progress interface to Twitter streaming API

**Usage**

```
twitterStreamer()
```

---

twitterTerms	<i>make a corpus object from results of a twitter REST search</i>
--------------	---

---

**Description**

All of the attributes returned by the twitterR library call are included as attributes in the corpus. A oauth key is required, for further instruction about the oauth processs see: <https://dev.twitter.com/apps/new> and the twitterR documentation

**Usage**

```
twitterTerms(query, numResults = 50, key, cons_secret, token, access_secret)
```

**Arguments**

query	Search string for twitter
numResults	Number of results desired.
key	Number of results desired.
key	'your consumer key here'
cons_secret	'your consumer secret here'
token	'your access token here'
access_secret	'your access secret here'

## Examples

```
## Not run:
twCorp <- twitterTerms('example', 10, key, cons_secret, token, access_secret)

## End(Not run)
```

wordfishMCMC

*Bayesian-MCMC version of the "wordfish" Poisson scaling model*

## Description

wordfishMCMC implements a flexible, Bayesian model estimated in JAGS using MCMC. It is based on the implementation of wordfish from the austin package. Options include specifying a model for alpha using document-level covariates, and partitioning the word parameters into different subsets, for instance, countries.

## Usage

```
wordfishMCMC(dtm, dir = c(1, 2), control = list(sigma = 3, startparams =
  NULL), alphaModel = c("free", "logdoclength", "modelled"),
  alphaFormula = NULL, alphaData = NULL, wordPartition = NULL,
  betaPartition = FALSE, wordConstraints = NULL, verbose = TRUE,
  PoissonGLM = FALSE, nChains = 1, nAdapt = 100, nUpdate = 300,
  nSamples = 100, nThin = 1, ...)
```

## Arguments

dtm	The document-term matrix. Ideally, documents form the rows of this matrix and words the columns, although it should be correctly coerced into the correct shape.
dir	A two-element vector, enforcing direction constraints on theta and beta, which ensure that $\theta_{\text{dir}[1]} < \theta_{\text{dir}[2]}$ . The elements of dir will index documents.
control	list specifies options for the estimation process. These are: tol, the proportional change in log likelihood sufficient to halt estimation, sigma the standard deviation for the beta prior in poisson form, and startparams a previously fitted wordfish model. verbose generates a running commentary during estimation. See <code>austin::wordfish</code> .
alphaModel	free means the $\alpha_i$ is entirely estimated; logdoclength means the alpha is predicted with an expected value equal to the log of the document length in words, similar to an offset in a Poisson model with variable exposure; modelled allows you to specify a formula and covariates for $\alpha_i$ using alphaFormula and alphaData.
alphaFormula	Model formula for hierarchical model predicting $\alpha_i$ .
alphaData	Data to form the model matrix for the hierarchical model predicting $\alpha_i$ .

wordPartition	A vector equal in length to the documents that specifies a unique value partitioning the word parameters. For example, alpha could be a Boolean variable for EU to indicate that a document came from a country outside the EU or inside the EU. Or, it could be a factor variable indicating the name of the country (as long as there are multiple documents per country). Internally, wordPartition is coerced to a factor. NULL indicates that no partitioning of the word-level parameters will take place (default).
betaPartition	Boolean indicating that the $\beta$ parameter should also be partitioned according to wordPartition.
wordConstraints	An index with a minimum length of 1, indicating which words will be set equal across the wordPartition factors. NULL if is.null(wordPartition) (default).
verbose	Turn this on for messages. Default is TRUE.
nChains	Number of chains to run in JAGS.
nAdapt	Adaptation iterations in JAGS.
nUpdate	Update iterations in JAGS.
nSamples	Number of posterior samples to draw in JAGS.
nThin	Thinning parameter for drawing posterior samples in JAGS.
PoissonGLM	Boolean denoting that the basic model should be estimated where $\log(\alpha)$ is $\sim \text{dflat}()$ as per The BUGS Book pp131-132
...	Additional arguments passed through.

**Value**

An augmented wordfish class object with additional stuff packed in. To be documented.

**Author(s)**

Kenneth Benoit

**Examples**

```
## Not run:
data(iebudgets)
# extract just the 2010 debates
iebudgets2010 <- corpus.subset(iebudgets, year==2010)

# create a document-term matrix and set the word margin to the columns
dtm <- create.fvm.corpus(iebudgets2010)
dtm <- wfm(t(dtm), word.margin=2)

# estimate the maximum likelihood wordfish model from austin
iebudgets2010_wordfish <- wordfish(dtm, dir=c(2,1))

# estimate the MCMC model, default values
iebudgets2010_wordfishMCMC <- wordfishMCMC(dtm, dir=c(2,1))
```

```
# compare the estimates of  $\theta_i$ 
plot(iebudgets2010_wordfish$theta, iebudgets2010_wordfishMCMC$theta)

# MCMC with a partition of the word parameters according to govt and opposition
# (FF and Greens were in government in during the debate over the 2010 budget)
# set the constraint on word partitioned parameters to be the same for "the" and "and"
iebudgets2010_wordfishMCMC_govtopp <-
  wordfishMCMC(dtm, dir=c(2,1),
    wordPartition=(iebudgets2010$attribs$party=="FF" | iebudgets2010$attribs$party=="Green"),
    betaPartition=TRUE, wordConstraints=which(words(dtm=="the")))

## End(Not run)
```

# Index

as.DocumentTermMatrix, [13](#)  
austin, [22](#)  
  
bigrams, [2](#)  
  
clean, [3](#)  
collocations, [4](#)  
corpusAddAttributes, [5](#)  
corpusAppend, [5](#)  
corpusCreate, [6](#)  
corpusFromFileNames, [6](#)  
corpusFromHeaders, [7](#)  
corpusReshape, [8](#)  
countSyllables, [8](#)  
create.fvm.corpus, [9](#)  
  
describeTexts, [10](#)  
dfm, [10](#), [11–14](#)  
dfm2ldaformat, [12](#)  
dfm2tmformat, [13](#)  
dfmTrim, [13](#)  
DocumentTermMatrix, [13](#)  
  
flatten.dictionary, [14](#)  
  
getRootFileNames, [15](#)  
getTextDir, [16](#)  
getTextDirGui, [17](#)  
getTextFiles, [17](#)  
getWordStat, [18](#)  
getWordStatCSV, [18](#)  
  
ieAttribs, [19](#)  
ieTexts, [19](#)  
ieTextsHeaders, [19](#)  
  
kwic, [19](#)  
kwic2, [20](#)  
  
lda, [12](#)  
lda.collapsed.gibbs.sampler, [12](#)  
  
likelihood.test, [21](#)  
  
MCMCirtPoisson1d, [22](#)  
movies, [24](#)  
  
naiveBayesText, [24](#)  
ngrams, [25](#)  
  
predict.naivebayes, [26](#)  
  
readWStatDict, [27](#)  
removeStopwords, [27](#)  
  
selectFeatures, [28](#)  
sentenceSeg, [29](#)  
simple triplet matrix, [13](#)  
stopwords\_EN, [11](#), [30](#)  
subset.corpus, [30](#)  
summary.corpus, [31](#)  
swissdebates, [31](#)  
sylCounts, [32](#)  
  
tagPos, [32](#)  
tokenize, [33](#)  
translate, [33](#)  
translate.corpus, [34](#)  
twitterSearch, [35](#)  
twitterStreamer, [35](#)  
twitterTerms, [35](#)  
  
wfm, [11](#)  
wordfish, [22](#), [23](#)  
wordfishMCMC, [36](#)